Joseph Pallechio

Lucas Harlor

KWIC Analysis

To tackle this assignment, we broke the first design down into classes. We went the route of development driven development. Started with the Line class and ensured that it could successfully pull a line and circular shift the first word to the end. The line class also overrides the compareTo and equals method in order for the LinkedList to sort the lines. Once that was completed we developed the array that will contain the linked list of each line. We struggled in getting the linked list to order, because our compareTo was not working as fast as expected; same with our toString. Once these hurdles were completed, our program could run. Our classes did rely on one another more than they should be, so we hoped to tackle that problem with our second design. In terms of the second design, we considered how the program would run if we had to change the UI design of the program. What if we needed it to run through a GUI and not console? Our first design had a lot of logic running through main and if we were to add a GUI component it would require us to just copy code straight from Main. We wanted to move towards a design that would allow both, without having to copy code. Our second design is more resilient to change. There is some data sharing between our classes, mostly for less time complexity, however our classes don't rely on each other. The only class that other classes are dependent on is the Line class, and it's only dependent on the add word method. If that method isn't change, for the worse, the program can easily be changed and have features added to it. Currently it is a console program, however, Main is very short and is only calling other methods in other classes. It could easily be made into a GUI program or a web based program. The classes would remain the same, just how they were used would be different. Both of our designs contain a file reader and writer method. If we were to change to a database, we would have to have another class that connects to the database and queries. This would affect how main functions significantly. In terms of our second design, main returns the file content. This could be adaptable to the database and wouldn’t be as much of a problem. Our first design is not as accustom to change as the second design. In the first design everything is handles in main. The GUI would have to handle reading the file, printing the file, sort and shift, and the ArrayList of Lines. In the second design, the GUI would have to handle the Arraylist of lines. An object called Lines could be created that would handle ArrayList<Line>, instead of having main or the GUI handle this. And then the GUI, console program, or web all could call that class when it needs to know something about the lines. The second design is easier to understand because each method is just doing one thing. This helps for the readability of the code and can be easily read by a developer. As stated in “Clean Code A Handbook of Agile Software Craftsmanship” by Robert Martin, each method should focus on doing one thing. Each method is also three lines or less as well. Though the second design is coded better, the performance is about the same. Unlike our first design, our second design follows the principle of information hiding. Our first design classes rely heavily on one another, where as our second design does not. Therefore, the second design can be easily adapted to a database or GUI. The second design is constructed with this capability in mind. The only class that is depended on is the Lines class meaning that the GUI can operate on its own without having to worry about the data in the back end.